Abductive Inference using Array-Based Logic

Jeppe Revall Frisvad[1] **Peter Falster**[1] **Gert L. Møller**[2] **Niels Jørgen Christensen**[1]

[1] Informatics and Mathematical Modelling
Technical University of Denmark
{jrf, pfa, njc}@imm.dtu.dk

[2] Array Technology A/S
P.O. Box 104, DK-1051 Copenhagen, Denmark
glm@arraytechnology.com

Abstract

The notion of abduction has found its usage within a wide variety of AI fields. Computing abductive solutions has, however, shown to be highly intractable in logic programming. To avoid this intractability we present a new approach to logic-based abduction; through the geometrical view of data employed in array-based logic we embrace abduction in a simple structural operation. We argue that a theory of abduction on this form allows for an implementation which, at runtime, can perform abductive inference quite efficiently on arbitrary rules of logic representing knowledge of finite domains.

1 Introduction

Abduction is used in logic only to translate the Greek word ἀπαγωγή (apagögé), which appears as a third kind of reasoning in Aristotle's Prior Analytics (Book II, Chap. XXV), see eg. the annotated translation in [Peirce, 1958, §§249–252] or the translation by A. J. Jenkinson which is available online[1]. Note, however, that ἀπαγωγή is translated by "reduction" in the latter version.

The logician Charles Sanders Peirce (1839–1914) was presumably the first to describe abduction as "the operation of adopting an explanatory hypothesis" [Peirce, 1960, §189]. He recognized this as a third kind of inference and he often argued that his theory was in agreement with that of Aristotle.

Other logicians mainly concentrate their efforts on the two other modes of inference: Deduction and induction. Little, if anything at all, was done about abduction until Chapter IV in [Hanson, 1958] revived Peirce's writings to inspire people such as Herbert Simon.

A mechanization of abductive logic (or hypothesis generation) to allow for computational approaches was first presented theoretically in [Morgan, 1971] and to the area of diagnosis in [Pople, 1973; Charniak and McDermott, 1985]. Charniak and McDermott also proposed natural language understanding as an area of application and areas as diverse as planning [Eshghi, 1988], default reasoning [Poole, 1988], knowledge assimilation [Kakas and Mancarella, 1990], scheduling [Kakas and Michael, 1999], and multiagent systems [Kowalski and Sadri, 1999] have subsequently been proposed. Only some of the first contributions to the different areas have been listed here. Several additional references and more applications can be found in [Kakas *et al.*, 1998; Magnani, 2001; Denecker and Kakas, 2002].

As noted in [Denecker and Kakas, 2002] many abduction-based systems have been developed, but few have been of "industrial scale". An exception is the air-crew scheduling application described in [Kakas and Michael, 1999]. There are several reasons why this is so: (a) It has been shown in [Eiter and Gottlob, 1995] that "even the simplest form of abduction is harder than deduction", (b) the systems implementing abductive reasoning most often achieve abduction through interaction with an inference engine build and optimized solely for deductive reasoning, and (c) they must always spend computational power at runtime to ensure consistency.

We cannot work around the fact (a), but we argue in the following sections that array-based logic combined with the main contribution of this paper - a structural operation for abduction - can improve on (b) and (c). The first since our operation can be worked at the core of the inference engine, the second since consistency is ensured as a part of the preprocessing in array-based logic (meaning that it is not necessary to ensure it at runtime).

2 Array-Based Logic

An analogy of Boolean logic "with coordinate-geometry and its invariant-theoretic foundations" [Mautner, 1946, p. 345] was first proposed by Mautner. He introduced the idea of a many-dimensional *logical coordinate system*, ie. a discrete cartesian coordinate system where each axis represents a Boolean variable, and thereby he connected Boolean logic to the mathematical group of geometrical transformations.

This point was further developed and given an operational form in [Franksen, 1979] where it ia. is shown that (a) projection in a logical coordinate system can "prove the theorems of divalent logic by computation" (projection is described in sec. 2.1), (b) an outer product can construct the relation between two variables on matrix form, and (c) "the operation of putting indices equal, is the operational implementation of repeated propositions in a propositional function" (ie. colligation, see sec. 2.2).

An illustration of the concept is given in figure 1 where a relation between two Boolean variables, pointed out as truth

[1]Eg. at http://etext.library.adelaide.edu.au/a/aristotle/a8pra/

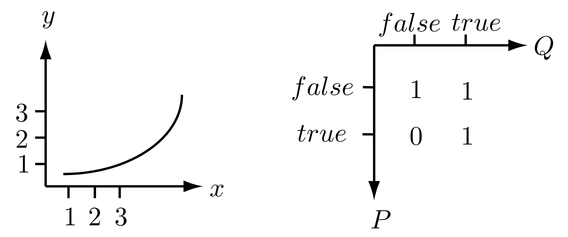

Figure 1: Comparison of the cartesian space spanned by two real variables $x, y \in \mathbb{R}$ and two Boolean variables $P, Q \in \mathbb{B}$. The relation between the variables is pictured as a curve chosen at random in the case of x and y and as the truth table of implication in the case of P and Q.

values in a two-dimensional logical coordinate system, can be compared to a curve in the traditional two-dimensional cartesian coordinate system. This indicates that in order to store a relation between n Boolean variables $(P_1, P_2, \dots, P_n)$ we must construct a data type of n dimensions. The array is a convenient choice and logic, in this sense, is, therefore, referred to as *array-based logic* (ABL).

Such a relation, stored in a many-dimensional array, consists of a Boolean value for each possible combination of values that the variables $(P_1, \dots, P_n \in \mathbb{B})$ can attain. A particular combination of values is, in fact, an index into the array and it is called a *configuration*. The Boolean value found at a certain index states whether a particular configuration is valid or not. An array spanning all the variables present in a system is referred to as the *system array*.

Through a generalization of the fundamental operations, enabling them to operate on many-dimensional arrays, ABL has been developed to its present state in [Franksen, 1996; 1997; Franksen and Falster, 2000; Møller, 1995; Pedersen, 1992]. In the sections 2.1, 2.2, and 2.3 we will give the definitions necessary for an implementation of ABL in an arbitrary programming language.

The gain from ABL as compared to the more traditional approaches based on manipulation of symbols (or searches in graphs) is that, when the system array has been constructed, we can not arrive at invalid or contradicting conclusions since all possible conclusions are accounted for. This makes inference at runtime very efficient. Deduction is merely a matter of splitting out subspaces or simply doing table look-ups. This means that the computational complexity of deduction on a system array is $O(1)$. The trade-off is obviously the considerable size of the array. Luckily there are options for compression, see eg. [Møller, 1995].

Structural operations, such as the outer product, are often done according to a Boolean function. The structural part of the operation is often general and can be defined as an operator taking a function[2] as argument and returning a new function. To account mathematically for such operators, or *transformers* as we prefer to call them, we employ the functional notation described in *array theory*. Array theory and many of the functions described in the following were developed by Trenchard More [1973; 1979].

[2]Sometimes more than one.

Image	State	Term
0 1	True	Affirmation
1 0	False	Negation
1 1	Indefinite	Tautology
0 0	Impossible	Contradiction

Table 1: Different possible one-dimensional arrays representing all possible values of a single Boolean variable.

The notation of array theory assumes left associativity, it also employs *currying* (meaning that the expression $A\, f$, where A is an array of data and f is a binary first order function, results in an unary function where A is bound to the first argument of f) and composition ($f\, g = f \circ g$).

2.1 Projection

An important property which our arrays must subsume is *nesting*. Consider n arrays of data: $A_0, \dots, A_n$. A successive juxtaposition of such arrays ($C = A_0 \, \dots \, A_n$) results in a single nested array (C) with its first element being A_0, its second element being A_1, etc. up to A_n. This is the only exception to the rule of left associativity and, if all the arrays are not just a single value, it is an example of nesting. Nesting is especially important when we need to split out axes from an array, see def. 1.

Definition 1 (split) *Let A be a list of numbers corresponding to axes in B, and let B be an array of data, then*

$$A \text{ split } B$$

returns a new array which has a nesting of the axes given in A. The nesting is placed next to the top level in B.

Suppose we want to split out P in the array illustrated in fig. 1, this corresponds to a nesting of the columns in the array:

$$\text{split}(0, A_{\text{sys}}) = 0 \text{ split } A_{\text{sys}} = \boxed{1\,0}\boxed{1\,1}$$

where A_{sys} is the system array representing implication between P and Q. An $\vee$-reduction on each nested element results in a new array specifying the relation between the variables that were not split out. Geometrically the split followed by an $\vee$-reduction corresponds to a *disjunctive projection* of the axes given as the first argument on the remaining axes of the array. Logically the disjunctive projection eliminates variables by the *principle of excluded middle* and leaves the relation between the remaining variables.

An $\vee$-reduction on the nested array given above results in the following:

$$A'_{\text{sys}} = 1\,1$$

which leads us to a discussion of the one-dimensional array representing all possible values of a single Boolean variable, that is, the *image* of a single variable. In this case the image of Q (which is the remaining variable in A'_{sys} after the disjunctive projection of P) states that Q may be either true or false. In other words it is *indefinite*. Table 1 lists the different possible images and their meaning with respect to the variable that they represent.

Operating a particular function on each nested element of an array can be defined as a transformer, see def. 2.

Definition 2 (EACH) *Let A be an array of data and let f be an unary first order function, then*

$$\text{EACH}\, f\, A$$

is defined as the function f applied to each element of the array A.

If we define that unary $\vee$ applied to an array of logic is the reduction transform of the binary $\vee$ (such as $\bigvee$, see the appendix of [More, 1979]) and, therefore, works between all elements in the array, an $\vee$-reduction corresponds to the function $\text{EACH} \vee$.

Having the transformer EACH and the function split we can construct a general geometrical operator (a transformer), which corresponds to *projection in a many-dimensional logical coordinate system according to an arbitrary reduction transform of a first order function f*:

$$\text{PROJECT}\, f = \text{EACH}\, f\, \text{split}$$

Now the image (and state) of a particular variable can be found by a disjunctive projection ($\text{PROJECT} \vee$) of all the other axes in the array.

2.2 Colligation

In order to construct a system array we must be able to construct the cartesian space spanned by several variables. The construction of such a space lies in the definition of a cartesian product, see def. 3.

Definition 3 (cart) *Let A be an m-dimensional array of data and let B be an n-dimensional array of data, then*

$$C = A\, \text{cart}\, B$$

returns an $(m+n)$-dimensional array, C, which holds the cartesian product of A and B. Meaning that C contains all possible pairs $a\, b$, where a is an element of A and b is an element of B. The first m axes of C corresponds to the axes of A and the last n axes of C corresponds to the axes of B.

Invoking an arbitrary binary first order function on each of the pairs resulting from a cartesian product corresponds to the general notion of an *outer product*. Hence, we can define *the outer product according to an arbitrary binary first order function* as follows:

$$\text{OUTER}\, f = \text{EACH}\, f\, \text{cart}$$

Using the outer product we can, now, construct our system arrays from rules written in classic propositional logic. The simple relation $P \Rightarrow Q$ pictured in fig. 1 is given as follows:

$$A_{\text{sys}} = \mathbb{B}\,(\text{OUTER} \Rightarrow)\,\mathbb{B} = \begin{matrix} 1\,1 \\ 0\,1 \end{matrix}$$

where the set of Boolean numbers, $\mathbb{B}$, in this context is regarded to be the ordered one-dimensional array: $0\,1$. The axes of the resulting array are enumerated such that axis 0 corresponds to P and axis 1 corresponds to Q. It is quite important to keep track of the axes and their affiliations. This is not stated explicitly since the axes are an inherent part of the array data structure.

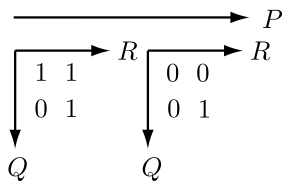

Figure 2: The Boolean variables corresponding to the different axes of C_{sys}.

Suppose a rule base is composed of two rules:

$$\begin{array}{ccc} P & \Rightarrow & Q \\ Q & \Rightarrow & R \end{array}$$

We could construct this rule base from the bottom using $\mathbb{B}$, but we could also build on A_{sys} found just before:

$$C'_{\text{sys}} = A_{\text{sys}}\,(\text{OUTER} \wedge)\,A_{\text{sys}} = \begin{matrix} 1\,1 & 1\,1 \\ 0\,1 & 0\,1 \\ & \\ 0\,0 & 1\,1 \\ 0\,0 & 0\,1 \end{matrix}$$

The system array C'_{sys} consists of four enumerated axes: $0\,1\,2\,3$ corresponding to the Boolean variables $P\,Q\,Q\,R$. This construction is inexpedient; there is no reason to have two equivalent axes.

In order to fuse the two equivalent axes we must pick out the elements where the two variables are equal (by putting indices equal). Geometrically this means that we must pick out the diagonal hyperplane between them. This process is referred to as *colligation*, see [Franksen and Falster, 2000].

Definition 4 (fuse) *Let A be a possibly nested list holding all the numbers corresponding to axes in B, and let B be an array of data, then*

$$C = A\, \text{fuse}\, B$$

arranges the axes of B according to the list provided in A such that the diagonal between the axes contained in each element of A becomes exactly one axis in the resulting array, C. If only one axis is specified as an element of A, the entire axis will be an axis in B. A must account for all axes in B and C must have exactly one axis for each element of A.

Colligation of axes 1 and 2 both representing Q in C'_{sys} is done, according to def. 4, as follows:

$$C_{\text{sys}} = 0\,(1\,2)\,3\; \text{fuse}\, C'_{\text{sys}} = \begin{matrix} 1\,1 & 0\,0 \\ 0\,1 & 0\,1 \end{matrix}$$

where the enumerated axes of C_{sys} are: $0\,1\,2$ corresponding to $P\,Q\,R$. The resulting array is exactly the truth table of the logical expression $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ only presented on array form. Figure 2 shows how the axes should be drawn in C_{sys}. Axis 0 is always the outermost axis.

In this way, using the outer product and colligation of axes, it is possible to construct a system array from an arbitrary set of rules. All logical operations describable as a binary first order function can be applied in the rule set. To keep the size of the resulting array from going out of bounds during the construction, colligation should be done whenever two or more equivalent axes appear.

As a short remark before we describe deductive inference on an array, it should be noted that a disjunctive projection of Q in C_{sys} results in the following relation between P and R:

$$1\ (\text{PROJECT} \vee)\, C_{\text{sys}} = \begin{matrix} 1\,1 \\ 0\,1 \end{matrix}$$

which states that $P \Rightarrow R$. This is, in fact, a geometrical proof of the hypothetical syllogism.

2.3 Deduction

In order to obtain the result of deductive inference on a system array (corresponding to a rule base) all we have to do is to pick out a subspace or do a simple table look-up.

If we need to test whether a given state vector is a possible configuration according to the current rule base, the table look-up is the relevant option. Testing whether $(P, Q, R) = (true, false, true)$ is possible according to the rule set of C_{sys} is simply done as follows:

$$1\,0\,1\ \text{pick}\, C_{\text{sys}} = 0$$

where pick is given as described in def. 5. The table look-up can be checked in fig. 2.

Definition 5 (pick) *Let A be a list of indices and let B be an array of data, then*

$$A\ \text{pick}\, B$$

picks the value in the array B at the position given by the indices in A. The number of indices in A must correspond to the number of axes in B.

Picking out a subspace can be done by the principle of nesting. A nesting of all axes, which do *not* correspond to variables that are bound to a specific value, can make sure that the relevant subspace is available by a simple pick.

In C_{sys} we can nest the axes: $1\,2$ corresponding to $Q\,R$. The result has only one axis corresponding to P, but each element is a nested two-dimensional array with the axes that were split out:

$$D = 1\,2\ \text{split}\, C_{\text{sys}} = \begin{array}{|c|c|} \hline 1\,1 & 0\,0 \\ 0\,1 & 0\,1 \\ \hline \end{array}$$

Using pick we can choose the subspace which holds the relation between Q and R as the result of P being either $false$ (0) or $true$ (1):

$$0\ \text{pick}\, D = \begin{matrix} 1\,1 \\ 0\,1 \end{matrix} \quad , \quad 1\ \text{pick}\, D = \begin{matrix} 0\,0 \\ 0\,1 \end{matrix}$$

which means that if P is $false$ the deduction results in the relation that $Q \Rightarrow R$, while if P is $true$ the resulting relation is $Q \wedge R$.

In the case where P is set true it should be noted that *modus ponens* is included indirectly in this simple geometrical operation which picks a subspace.

As long as the system array is colligated and specifies the correct relation between the variables that its axes represent, the picking of a subspace will include all kinds of deductive inference. This is true since the array encompass all possible configurations.

3 Abduction

As an extension of ABL we will in the following present a new structural operation for abduction and describe the type of abduction that it models with references to Peirce.

Reading Aristotle it seems reasonable to assume that abduction is a way of establishing a *credible* hypothesis, which leads to a *desired* conclusion. The following is our symbolic interpretation of Aristotle's ἀπαγωγή:

$$\begin{array}{ccc} P & \Rightarrow & Q \\ Q' & \Rightarrow & R \\ \hline P & \Rightarrow & R \end{array}$$

The desired conclusion is in this case that $P \Rightarrow R$. A credible hypothesis might be (a) that $Q = Q'$ or (b) that $Q \Rightarrow Q'$ or even (c) that $(Q \Rightarrow Q'') \wedge (Q'' \Rightarrow Q')$ and so forth. Aristotle argues that a hypothesis can be accepted abductively if it is "more credible than is the conclusion and if moreover the middles between the middle and the minor term be few". In (a) there are no middles between the middle and the minor term, in (b) there is one, in (c) there are two, etc.

Recall from sec. 2.2 that the syllogism is a direct consequence of colligation. The different cases could be handled by colligation of two axes (case (a)) or by addition of new rules to the system array (other cases). The credibility of a particular hypothesis is, however, not obvious from an operational perspective.

It is interesting to note that Aristotle in the form of syllogisms always works with logical conclusions that are internally present in the rule set. For conclusions based on external influences we must turn to the Stoic school of logic and their *modi*. In sec. 2.3 it was shown how this kind of reasoning is modeled by the picking of a subspace.

Peirce formulated the notion of abduction in terms of the Stoic modi as follows, [Peirce, 1960, §189]:

> The surprising fact, C, is observed;
> But if A were true, C would be a matter of course,
> Hence, there is reason to suspect that A is true

The general idea in Peirce's theory of abduction is, as we see it, in agreement with that of Aristotle. The difference is that Peirce's version seeks external influences leading to the desired conclusion (or *the observed fact* in Peirce's terminology), while Aristotle's version seeks changes in the rule set leading to the desired conclusion.

Since Peirce's abduction does not seek to change the rule set, but rather to find hypotheses that according to the rule set leads to an observed fact, Peirce's abduction is readily suitable for an operational approach.

In a discussion about the differences between induction and abduction Peirce gives the following statements [Peirce, 1958, §218]:

> Abduction makes its start from the facts [...] the consideration of the facts suggests the hypothesis. [...] The mode of suggestion by which, in abduction, the facts suggest the hypothesis is by *resemblance*, – the resemblance of the facts to the consequences of the hypothesis.

If we strain Peirce's description of abduction a little and substitute the word *resemblance* by the stronger word *equality*, we have a "strong case" of abduction for which we can construct an operation, see def. 6. The operation consists in a nesting of the axes corresponding to the observed variables and afterwards a pattern match for equality on each nested element.

Definition 6 (abduct) *Let A be an array of data representing an observed fact, let B be a list of numbers such that element i of B tells which axis in C axis i in A corresponds to, and let C be an array of data then the "strong case" of abduction is given as:*

$$D = \mathrm{abduct}(A, B, C) = B \ \ (\mathrm{PROJECT}\,(A =)) \ \ C$$

where the relation in D (between the variables corresponding to axes remaining after the projection) constitutes the hypothesis.

Abduction, as given by the abduct operation, is exactly opposite to picking a subspace. After finding that P is true we can *deduce* $Q \wedge R$ by a nesting of the axes corresponding to R and Q in sec. 2.3. In *abduction* we could observe that R is $true$, then the fact is $A_1 = 0\,1$ (cf. tab. 1) and $B_1 = 2$ since the first axis in A_1 corresponds to axis 2 in C_{sys}. The nesting of axis 2 in C_{sys} has the following result:

$$D_1 = B_1 \ \mathrm{split}\, C_{\mathrm{sys}} = \begin{array}{|c|c|}\hline 1\,1 & 0\,1 \\ \hline 0\,0 & 0\,1 \\ \hline \end{array}$$

and the subsequent match of equality between observed fact and nested elements results in the following hypothesis:

$$\mathrm{abduct}(A_1, B_1, C_{\mathrm{sys}}) = \mathrm{EACH}\,(A_1 =)\ D_1 = \begin{array}{c} 0\,1 \\ 0\,1 \end{array}$$

where the remaining axes correspond to P and Q. It is evident (by disjunctive projection of the opposite axis) that P is indefinite and Q is bound to truth. Our abduction could end at this point. On the other hand Q has, abductively, been rendered $true$ and we might as well iterate over the operation and make an abduction on the result: $Q \wedge R$. Observe that this second iteration is the exact opposite of the example in sec. 2.3. The fact is now:

$$A_2 = \mathbb{B}\ (\mathrm{OUTER} \wedge)\ \mathbb{B} = \begin{array}{c} 0\,0 \\ 0\,1 \end{array}$$

while $B_2 = 1\,2$ and the nested array $D_2 = D$ of sec. 2.3. The result of the second iteration is:

$$\mathrm{abduct}(A_2, B_2, C_{\mathrm{sys}}) = 0\,1$$

where the remaining axis corresponds to P. Hence, the hypotheses resulting from the second iteration is simply that if P was $true$, R (and Q) would be a matter of course.

If the resulting hypothesis is a contradiction, that is, an array holding the value 0 at all positions, no configuration exists in the closed system which can lead to the observed fact.

We suggest two ways to handle the possibility of iteration: (a) It can continue till all variables represented in the hypothesis are either bound or indefinite, or (b) a set of *abducible* variables can be introduced as in abductive logic programming, see [Eshghi and Kowalski, 1989]. If we choose a set of abducibles, the iteration will continue as long as an abducible is bound to truth or falsehood in the hypothesis resulting from an iteration.

Since the truth values in a system array, say T, by definition points out consistent configurations only, the projective pattern match will find *all* the *consistent* configurations that, in the closed system, deductively renders the observation true. The array holding these configurations is the hypothesis, Δ, which only concerns the variables that have not been abduced. This means that the conjunction of the hypothesis and the system array, $T \wedge \Delta$, after colligation will point out configurations, concerning all variables in the system, which logically entails the conclusion. In other words:

$$T \wedge \Delta \Rightarrow Q$$

will always be a tautology. This ensures that our operation for abduction fulfils the *correctness criterion* for abductive reasoning, cf. [Denecker and Kakas, 2002].

The complexity of our operation for abduction is $O(nm)$, where n is the number of configurations in our array and m is the number of iterations. Note, however, that each iteration will abduce a larger amount of variables leaving fewer variables in the hypothesis, meaning that the process is, indeed, deterministic. This fact and the point that our inference is rid of searches in graphs and guarantied to uphold consistency without checking it at runtime makes us confident that it has a quite efficient implementation.

4 Discussion

An interpreted development language called Nial (Nested interactive language) was originally proposed in [Jenkins, 1981] for the purpose of testing array theoretic concepts. The theory presented in this paper can be tested immediately in Nial[3], since the nested array data structure as well as a compatible version of the functions and transformers given in definitions 1 through 5 are a part of the language.

An efficient tool making ABL available for large scale applications was based on [Møller, 1995]. It consists of an Array Compiler and an Array Runtime library.[4] The Array Compiler calculates the system array using the more compact *disjunctive normal form*. Meaning that only valid (or $true$) configurations are stored. To obtain further compression the system array is split into one or more nested arrays with each valid cartesian subspace given a unique index. This makes Array Runtime able to perform real-time deduction on a very compact representation, since the task is still basically indexing and table look-ups. To adapt our theory of abduction for Array Runtime a few changes must be made to account for the compact form. An adapted operation might, however, be more efficient since n in $O(nm)$ is potentially only the number of *valid* configurations.

The limited presentation of array-based logic given in this paper has set the stage for rules written in propositional logic. The same general operations can, however, also be used to construct system arrays based on rules written in predicate

[3]Available at http://www.nial.com/.

[4]See http://www.arraytechnology.com/

logic. This option is described in [Møller, 1995] and it is also a part of the Array Compiler/Runtime tool.

5 Conclusion

After a short description of ABL we have shown how a "strong case" of abduction can be defined as a relatively simple structural operation (see def. 6). This makes a new approach to logic-based abductive inference possible. Sufficient details for its implementation has been provided in this paper.

Through the employment of ABL we achieve an inference engine which, at runtime, need not perform expensive searches in graphs or spend computational power for consistency checks. The operations employed are predictable in storage requirements and processing time, which is important if logic-based reasoning is to be adopted in real-time systems.

The operation we present for abduction is still more complex than that of deduction, but it is deterministic, it consists only of simple equality checks, and it can not arrive at inconsistent hypotheses (see sec. 3). This makes it suitable for integration at the core of an inference engine. An iteration of the operation is possible to further limit the possible configurations in the hypothesis.

The limitation of our approach is that rule sets must represent knowledge of finite domains only since all possible configurations will be represented in the system array. The array is consequently of considerable size and techniques for compression must be employed to allow for "industrial scale" application. An implementation of our operation in Array Technology's Array Runtime library would overcome this issue.

References

[Charniak and McDermott, 1985] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, 1985.

[Denecker and Kakas, 2002] M. Denecker and A. Kakas. Abduction in logic programming. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 1. Springer, 2002.

[Eiter and Gottlob, 1995] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.

[Eshghi and Kowalski, 1989] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In *Proc. of the 6th International Conference on Logic Programming*, pages 234–255. MIT press, 1989.

[Eshghi, 1988] K. Eshghi. Abductive planning with event calculus. In R. A. Kowalski and K. A. Bowen, editors, *Proc. of the 5th International Conference on Logic Programming*, pages 562–579. MIT press, 1988.

[Franksen and Falster, 2000] O. I. Franksen and P. Falster. Colligation or, the logical inference of interconnection. *Mathematics and Computers in Simulation*, 52(1):1–9, March 2000.

[Franksen, 1979] O. I. Franksen. Group representation of finite polyvalent logic: A case study using APL notation. In A. Niemi, editor, *A Link between Science and Applications of Automatic Control, IFAC VII, World Congress 1978*, pages 875–887, Oxford, 1979. Pergamon Press.

[Franksen, 1996] O. I. Franksen. Invariance under nesting - an aspect of array-based logic with relation to grassmann and peirce. In G. Schubring, editor, *Hermann Günther Graßmann (1809-1877): Visionary Mathematician, Scientist and Neohumanist Scholar*, pages 303–335, Dordrecht, 1996. Kluwer Academic Publishers.

[Franksen, 1997] O. I. Franksen. Boole's development process revisited: From an array-theoretic viewpoint. *Acta historica Leopoldina*, 27:175–188, 1997.

[Hanson, 1958] N. R. Hanson. *Patterns of Discovery: An Inquiry into the Conceptual Foundations of Science*. Cambridge University Press, 1958.

[Jenkins, 1981] M. A. Jenkins. A development system for testing array theory concepts. *ACM SIGAPL APL Quote Quad*, 12(1):152–159, September 1981.

[Kakas and Mancarella, 1990] A. C. Kakas and P. Mancarella. Knowledge assimilation and abduction. International Workshop on Truth Maintenance. In *Proc. of the European Conference on Artificial Intelligence*, volume 515 of *Lecture Notes in Computer Science*, pages 54–71. Springer, 1990.

[Kakas and Michael, 1999] A. C. Kakas and A. Michael. Air-crew scheduling through abduction. In *Proc. of IEA/AIE-99*, pages 600–612, 1999.

[Kakas *et al.*, 1998] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235–324. Oxford University Press, 1998.

[Kowalski and Sadri, 1999] R. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25:391–419, 1999.

[Magnani, 2001] L. Magnani. *Abduction, Reason, and Science: Processes of Discovery and Explanation*. Kluwer Academic/Plenum Publishers, New York, 2001.

[Mautner, 1946] F. I. Mautner. An extension of klein's erlanger program: Logic as invariant-theory. *American Journal of Mathematics*, 68(3):345–384, July 1946.

[Møller, 1995] G. L. Møller. *On the Technology of Array-Based Logic*. PhD thesis, Electrical Power Engineering Department, Technical University of Denmark, 1995. Available at http://www.arraytechnology.com/.

[More, 1973] T. More, Jr. Axioms and theorems for a theory of arrays. *IBM Journal of Research and Development*, 17(2):135–175, 1973.

[More, 1979] T. More, Jr. The nested rectangular array as a model of data. In *Proceedings of the International Conference on APL: Part 1*, pages 55–73, 1979.

[Morgan, 1971] C. G. Morgan. Hypothesis generation by machine. *Artificial Intelligence*, 2(2):179–187, 1971.

[Pedersen, 1992] A. Pedersen. *Digraph Representation in Array-Based Logic: With Special Emphasis on the Mathematical Foundation of Production Models*. PhD thesis, Electrical Power Engineering Department, Technical University of Denmark, 1992.

[Peirce, 1958] C. S. Peirce. On the logic of drawing history from ancient documents especially from testimonies (1901). In A. W. Burks, editor, *Collected Papers of Charles Sanders Peirce*, volume VII, book II, chap. 3. Harvard University Press, 1958.

[Peirce, 1960] C. S. Peirce. Lectures on pragmatism: Lecture VII (1903). In C. Hartshorne and P. Weiss, editors, *Collected Papers of Charles Sanders Peirce*, volume V, book I. Harvard University Press, second printing, 1960.

[Poole, 1988] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[Pople, 1973] H. E. Pople. On the mechanization of abductive logic. In *Proc. of the 3rd International Joint Conference on Artificial Intelligence*, pages 147–152, 1973.